Getting started with FOSS Data Science & Machine Learning Tools

Atlanta Developers' Conference
September 17, 2022

ATL•DEV•CON

9•17•2022

Platinum Sponsors

ScrumSimple.com

Scrum. Simple.

KENNESAW STATE UNIVERSITY

Gold Sponsors

epam

Microsoft

PILOT COMPANY

slalom

tyler technologies

Attendee Party Sponsor

improving

It's what we do. ™

Silver Sponsor

atmosera

Swag Sponsor

Kontent.ai

Who am I?

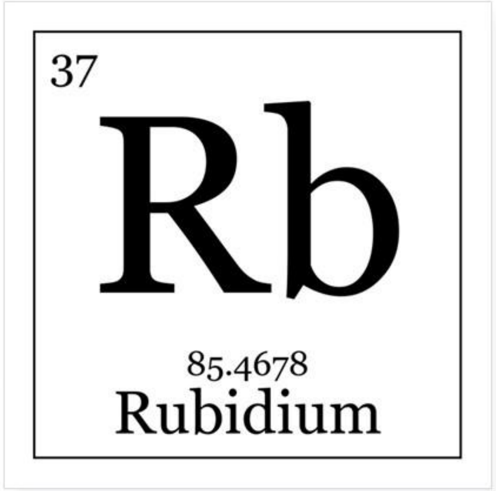
Robert Bates

Senior Backend Engineer
X-Force Threat Intelligence @ IBM

arpieb most everywhere online

What are my data science and machine learning qualifications?

- Studied Machine Learning and AI at Georgia Tech
- Research Faculty at GT in the Design & Intelligence Lab during grad school
- Instructional Associate for CS 7641 "*Introduction to Machine Learning*"
- Head IA + Course Developer for CS 8001-ODM "*Machine Learning and Data Science Tooling*"
- Application of data science and machine learning in industry at both Veloxiti AI and IBM Security



What is FOSS? Why is it important?

*"FOSS means **Free** and **Open Source Software**.*

It doesn't mean software is free of cost.

*It means that **source code** of the software is open for all and anyone is **free to use, study and modify** the code.*

*This principle allows **other people to contribute to the development and improvement** of a software like **a community**."*

– https://itsfoss.com/what-is-foss/

What is FOSS? Why is it important?

FOSS software is readily available and typically supported on multiple platforms (Windows, macOS, Nix-ish, mobile/embedded, etc).

Community nature means you can provided a fix as well as provide feedback and testing at a scale that most commercial ventures could never reach.

Want something slightly different or new? Fork it, contribute it back!

Many commercial systems are built on top of FOSS tools and technologies, so learning these building blocks often gives you an on-ramp to enterprise-class tools from the comfort of your own systems.

What are we going to cover?

There are some terrific, well-adopted and well-supported tools out there for beginning data scientists and machine learning practitioners.

There are also a lot of confusing, poorly-documented, and poorly-adopted tools with a lot of overlap.

Let’s take the happy path, yes?

😁

What are we going to cover?

Python 3 tools and frameworks because:

- Workhorse of most data science, machine learning, and NLP frameworks out there.
- Well-supported, documented, and available.
- Pick yer flavor - Anaconda (aka “Big Snake), Conda, Pip, Virtualenv, etc ad nauseum for setting up localized environments depending on your expertise and platform.
    - I’m a pip+virtualenv type, so the examples will be littered with pip-managed packages in a venv. No 🔥 please!
- Reasonably easy to add support for compiled, native, high-performance code as needed (e.g. numpy, scipy).

Yes, I’m aware there are tools for `<insert language of choice>` but Python is still the best place to dip your toes in the ocean IMHO. The core concepts will carry over.

(And you can get a lot done without a lot of boilerplate.)

What are we going to cover?

Project Jupyter - particularly the Jupyter Lab notebook framework

Handling tabular data with Pandas

Visualizing data with Seaborn + Matplotlib

Handling network/graph data with NetworkX

Natural Language Processing (NLP) with NLTK, GenSim, and spaCy

Intro to Machine Learning with Scikit-learn

IF we have time, brief look at Pytorch Lightning and Keras for Deep Learning

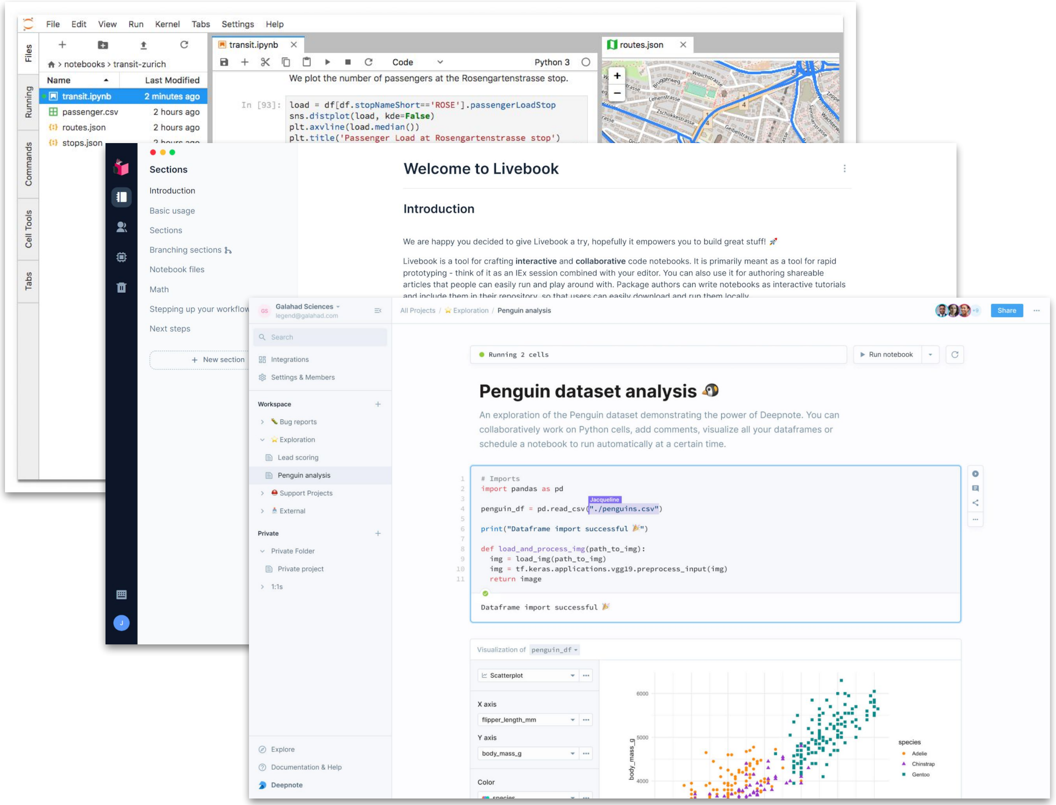
Jupyter Notebooks

No, not a journal about the planet…

What is a notebook?

A **notebook** in the context of DS/ML is a web-based application that blends **code**, **visualizations**, and **documentation** into **a single interactive document**.

Jupyter, DeepNote and LiveBook are all examples of notebook systems.



We plot the number of passengers at the Rosengartenstrasse stop.

```
In [93]: load = df[df.stopNameShort=='ROSE'].passengerLoadStop
         sns.distplot(load, kde=False)
         plt.axvline(load.median())
         plt.title('Passenger Load at Rosengartenstrasse stop')
```

Welcome to Livebook

Introduction

We are happy you decided to give Livebook a try, hopefully it empowers you to build great stuff! 🚀

Livebook is a tool for crafting **interactive** and **collaborative** code notebooks. It is primarily meant as a tool for rapid prototyping - think of it as an IEx session combined with your editor. You can also use it for authoring shareable articles that people can easily run and play around with. Package authors can write notebooks as interactive tutorials and include them in their repository, so that users can easily download and run them locally.

Penguin dataset analysis 🐧

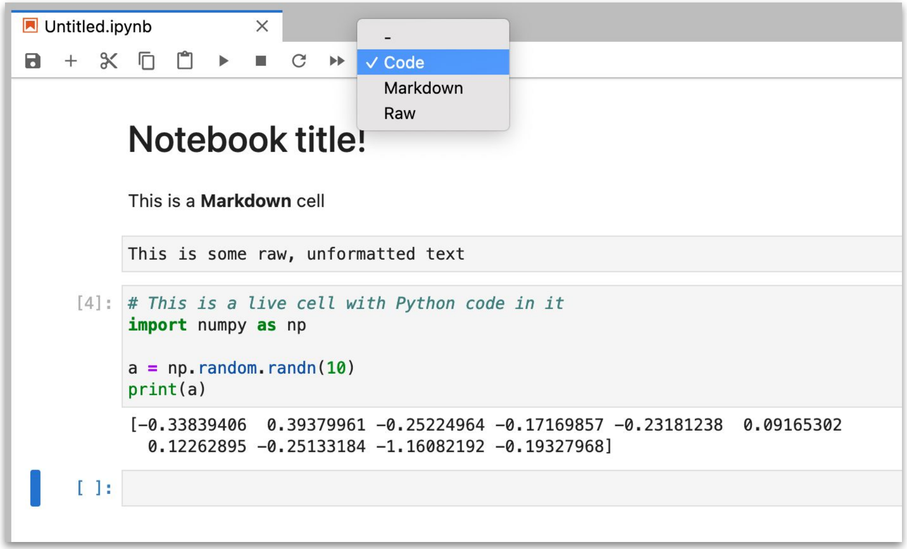
An exploration of the Penguin dataset demonstrating the power of Deepnote. You can collaboratively work on Python cells, add comments, visualize all your dataframes or schedule a notebook to run automatically at a certain time.

```
# Imports
import pandas as pd

penguin_df = pd.read_csv("./penguins.csv")

print("Dataframe import successful 🎉")

def load_and_process_img(path_to_img):
  img = load_img(path_to_img)
  img = tf.keras.applications.vgg19.preprocess_input(img)
  return image
```

```
Dataframe import successful 🎉
```

What is a Jupyter notebook?

A **Jupyter notebook** is a single-file document (.IPYNB extension) that contains one or more "cells" of content. Cells can contain Markdown, raw text, or code. Each cell is "executable" which means once you enter something into them, they can be run through the current kernel (in this example Python) and output can be captured within the document.



Notebook title!

This is a **Markdown** cell

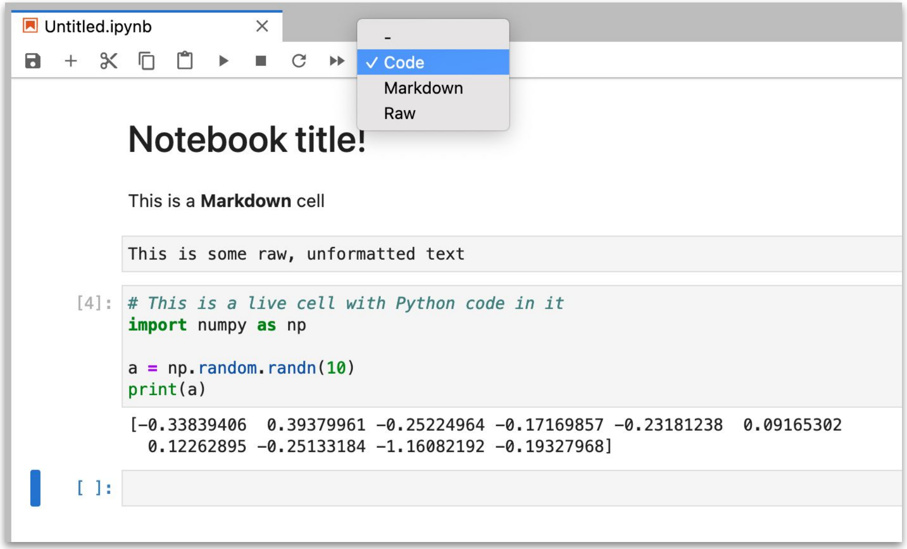
```
This is some raw, unformatted text
```

```
[4]: # This is a live cell with Python code in it
     import numpy as np

     a = np.random.randn(10)
     print(a)

     [-0.33839406  0.39379961 -0.25224964 -0.17169857 -0.23181238  0.09165302
       0.12262895 -0.25133184 -1.16082192 -0.19327968]

[ ]:
```

What is a Jupyter notebook?

The results of each cell are tracked in the current kernel so cells can leverage code and variables defined in the order they appear in the notebook, just like a script file.

In the example, the next cell could use the variable **a** that was defined above it, as well as access the imported packages.



Notebook title!

This is a **Markdown** cell

```
This is some raw, unformatted text
```

```
[4]: # This is a live cell with Python code in it
     import numpy as np

     a = np.random.randn(10)
     print(a)
```

```
[-0.33839406  0.39379961 -0.25224964 -0.17169857 -0.23181238  0.09165302
  0.12262895 -0.25133184 -1.16082192 -0.19327968]
```

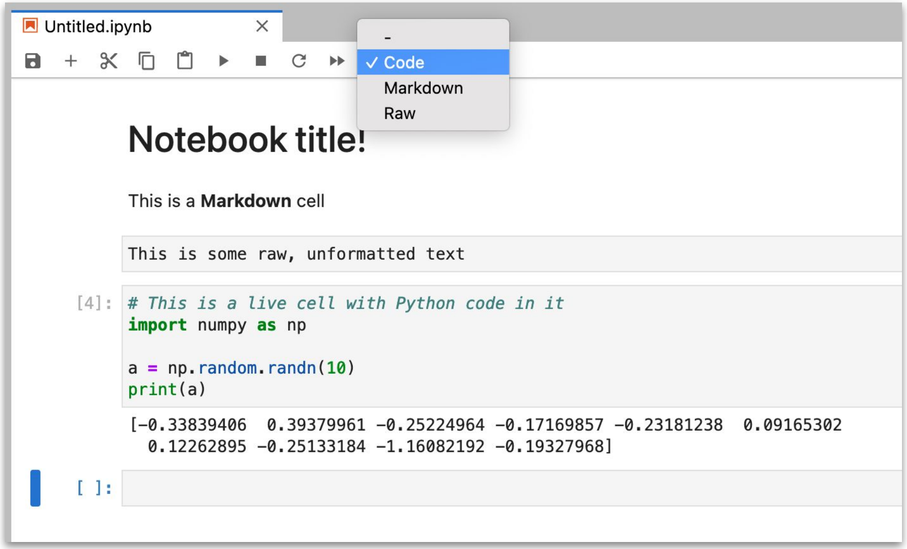
```
[ ]:
```

What is a Jupyter notebook?

Unlike a normal script, cells can be executed independently of each other, but if they modify global state (e.g. redefine functions, variables, imports, etc) they will affect all the rest of the cells executed after them.

This is one of the biggest gotcha's (and critiques) of many notebook implementations.

(Elixir's LiveBook is one of the few exceptions, but is beyond the scope of this presentation.)



Notebook title!

This is a **Markdown** cell

```
This is some raw, unformatted text
```

```
[4]: # This is a live cell with Python code in it
     import numpy as np

     a = np.random.randn(10)
     print(a)
```

```
[-0.33839406  0.39379961 -0.25224964 -0.17169857 -0.23181238  0.09165302
  0.12262895 -0.25133184 -1.16082192 -0.19327968]
```

```
[ ]:
```

Why use Jupyter notebooks in DS/ML?

Jupyter notebooks provide a rich interactive environment for performing rapid data exploration, analysis, visualizations, and code proof-of-concepts (PoC).

They are lighter weight and more intuitive than most programming language IDEs, making them ideal for users who might not be as technically inclined as a full-blown software engineer.

Jupyter notebooks can be collaborated on by multiple researchers via the web, and can be shared with other researchers in either IPYNB format or exported into several formats for publishing (e.g. PDF, LaTeX, HTML, etc).

Jupyter notebooks are supported on nearly every major DS/ML cloud platform as a starting point for building solutions for that platform (Azure, AWS, Google Cloud).

Jupyter, Jupyter Lab, or Jupyter Hub?

The original **Jupyter** package is being deprecated in favor of the newer implementation called **Jupyter Lab**. I would strongly suggest you use Jupyter Lab for your projects and going forward. The notebooks are interchangeable between the two implementations, and are still referred to as Jupyter notebooks.

There is another package called **Jupyter Hub** that is used to provide a hosted, multi-user environment for Jupyter Lab. I would strongly suggest you DON’T mess with this as setup and configuration is not for the faint of heart, and it’s overkill unless you’re trying to support an entire team of researchers with very specific requirements.

Handling tabular data

Datasets, DataFrames, and Files! (Oh my!)

Datasets, DataFrames and Data Files

Datasets are sources of data that have been created by researchers.

Data Files are how datasets are stored, and there are several common formats with their intended use cases, pros, and cons.

DataFrames are a standard abstraction for tabular data loaded from data files, which are provided by a dataset.

Data Files

Datasets are often made available in either a single file or collection of files organized either by folder name or some key embedded in the filename.

There are dozens of data formats out there, but there are a few that you will encounter way more often than others:

- Comma Separated Values (CSV, .csv)
- Apache Parquet (.parquet)
- Hierarchical Data Format (HDF, .hdf, .hdf5, .h5)
- Attribute-Relation File Format (ARFF, .arff)
- Javascript Object Notation (JSON, .json)

DataFrames

A **DataFrame** is a data structure that organizes data into a 2-dimensional table of rows and columns, much like a spreadsheet. DataFrames are one of the most common data structures used in modern data analytics because they are a flexible and intuitive way of storing and working with data.

https://databricks.com/glossary/what-are-dataframes

DataFrames are probably the most common, most useful, and best supported abstractions for tabular data you will run across in DS/ML applications. They are often optimized for columnar operations, but are equally efficient executing MapReduce operations across large numbers of rows.

DataFrames

In the Python ecosystem, **Pandas** is the go-to package for manipulating dataframes. It provides for the entire lifecycle of dataframe processing (reading, processing, plotting, and saving).

For a more powerful dataframe implementation, **Dask** provides support for dataframes that will not fit in memory, and can even distribute processing across multiple computers in a cluster, via a very Pandas-like API.

For truly big data systems, **PySpark** provides an interface to the Apache Spark dataframe implementation, which while somewhat different from Pandas, will look very familiar one you’ve got some experience with the dataframe paradigm.

Datasets

There are many public sources of datasets available; some have better metadata and documentation than others. Some commonly used datasets when learning DS and ML can be found at these public repositories or search engines:

- https://archive.ics.uci.edu/ml/datasets.php
- https://www.kaggle.com/datasets
- https://paperswithcode.com/
- https://www.datasetlist.com/
- https://index.quantumstat.com/
- https://data.gov/open-gov/
- https://datasetsearch.research.google.com/

Visualizing data

Why is data visualization ("dataviz") important?

Data visualization primarily comes in two forms:

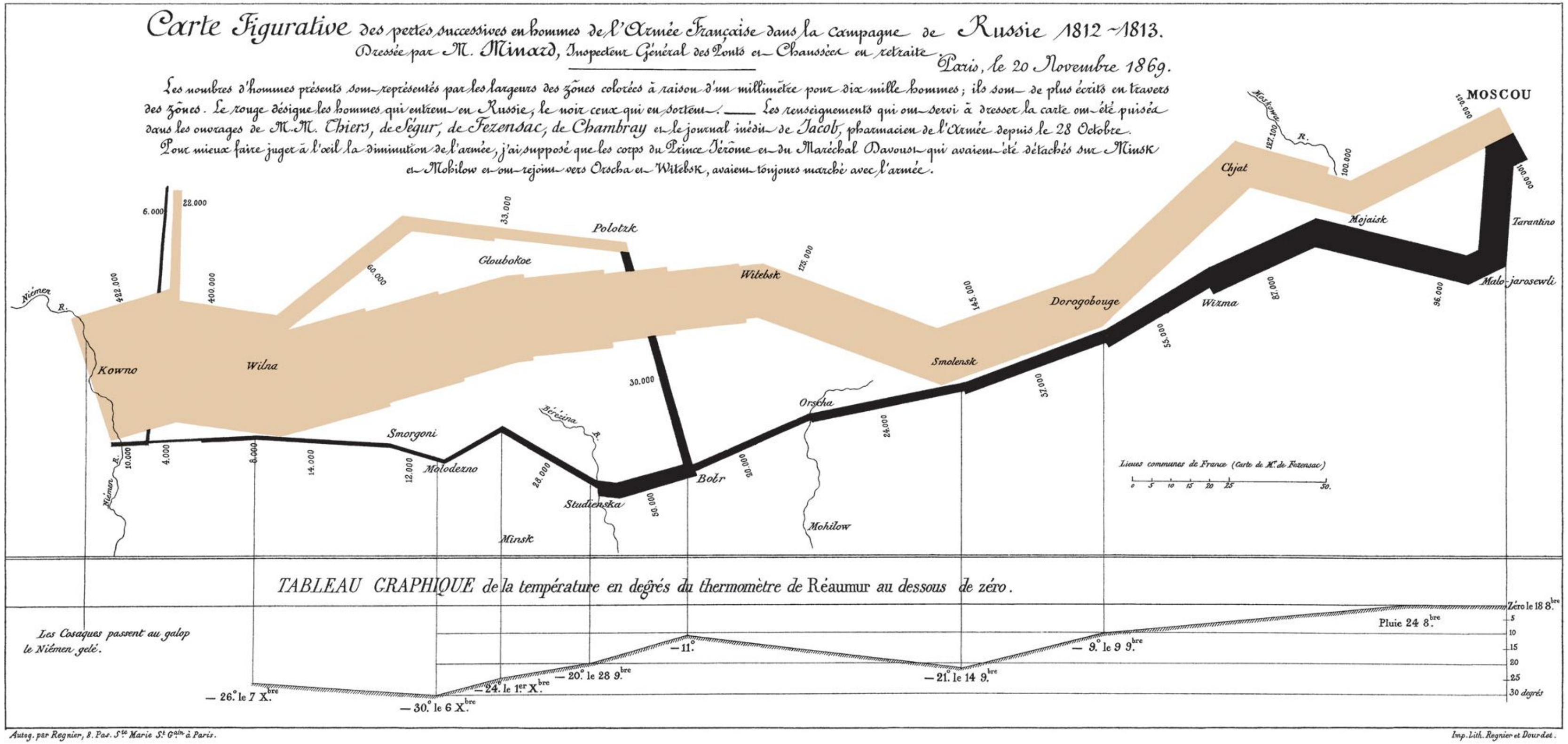
Analysis tool

- Allows data scientists and ML researchers to wrap their head around a large state space
- Surfaces trends/structures inherent in the data
- Rapid exploration of a dataset for exceptions, outliers, etc

Communications tool

- Stakeholders can easily see what you are trying to convey
- Reduces cognitive load for those not intimately familiar with the data
- Can provide a call to action

Quality of dataviz



Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812–1813.
Dressée par M. Minard, Inspecteur Général des Ponts et Chaussées en retraite. Paris, le 20 Novembre 1869.

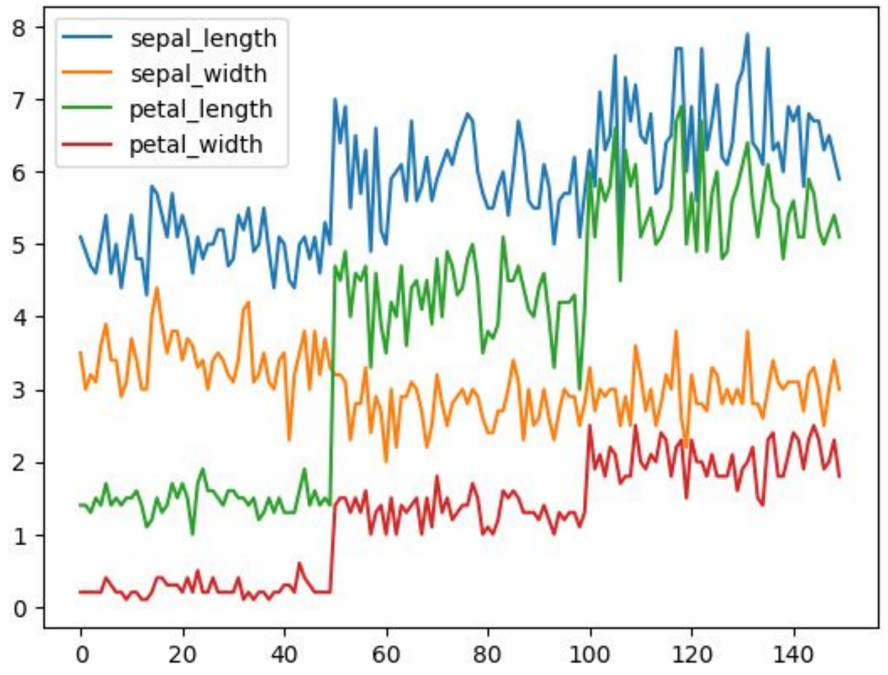
Les nombres d'hommes présents sont représentés par les largeurs des zônes colorées à raison d'un millimètre pour dix mille hommes; ils sont de plus écrits en travers des zônes. Le rouge désigne les hommes qui entrent en Russie, le noir ceux qui en sortent. —— Les renseignements qui ont servi à dresser la carte ont été puisés dans les ouvrages de MM. Thiers, de Ségur, de Fezensac, de Chambray et le journal inédit de Jacob, pharmacien de l'Armée depuis le 28 Octobre.
Pour mieux faire juger à l'œil la diminution de l'armée, j'ai supposé que les corps du Prince Jérôme et du Maréchal Davoust qui avaient été détachés sur Minsk et Mohilow et ont rejoint vers Orscha et Witebsk, avaient toujours marché avec l'armée.

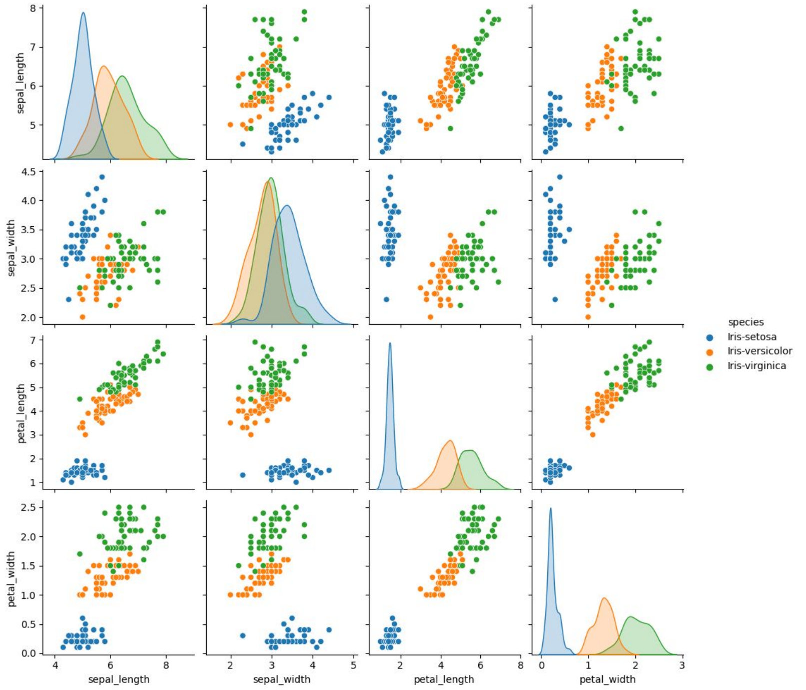
Quality of dataviz

Carte figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813 by Charles Joseph Minard.

The illustration is perhaps the single best-known **statistical graphic of the nineteenth century** and depicts Napoleon's army departing the Polish-Russian border. A thick band illustrates the size of his army at specific geographic points during their advance and retreat. **It displays six types of data in two dimensions**.

Both are from the same data...





Your tools can make this much easier

Matplotlib is the go-to for most seasoned data scientists, but it has a somewhat archaic interface and can be difficult to style or generate complex visualizations. Many people new to it find it to be challenging at best. (Bring the 🔥)

Seaborn brings robust abstractions and aesthetically pleasing default styles to Matplotlib while still allowing granular control if you really want/need it.

Plotly is also now quite popular for dataviz since they got away from, "Just upload your data to us, here's a JS API…" and has a rich set of visualizations to choose from.

Depending on your specific use case, there are often many other visualization packages that can provide very specific visualizations and/or infographics from raw data.

Handling network/graph data

It’s nodes all the way down

What are networks/graphs?

Network and **graph** are both incredibly overloaded terms:

- https://en.wikipedia.org/wiki/Network
- https://en.wikipedia.org/wiki/Graph

We are going to discuss them in the context of **data relationship representation**, where they are often used interchangeably (e.g. “professional *network*”, “social *graph*”, etc).

Even in this context, the term **graph** can sometimes be overloaded based on how the data structure is going to be used.

What are networks/graphs?

Graphs are **significantly different from tabular data**, which is what we’ve seen in datasets up until now.

They are used to model **relationships** and/or **interactions** between **entities/concepts**.

Graphs are becoming more prevalent as **graph theory** and **deep learning** are both making significant advances in analyzing complex graph structures, enabling **more insightful inferences on graph data**.

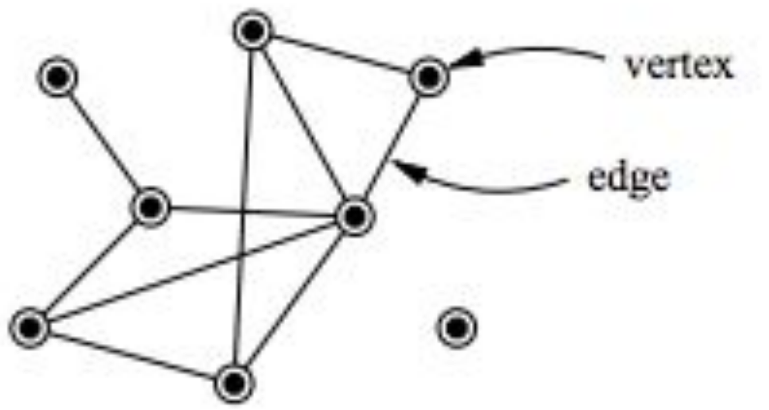
Core graph concepts

The core components in all graphs are the **vertex** (aka **node**) and **edge**.

A **vertex** represents a known datum that has zero or more relationships/interactions with other vertices.

An **edge** represents the relationship/interaction between a pair of vertices.

A **path** is a list of vertices and edges that describes a traversal between two vertices in a graph.



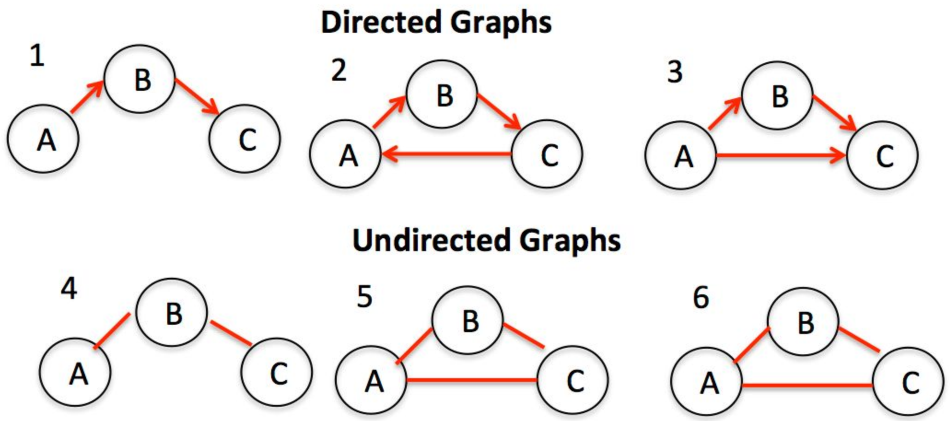
https://commons.wikimedia.org/wiki/File:Small_Network.png

Core graph concepts

Graphs can by any combination of directed or undirected, acyclic or cyclic:

- Directed acyclic graph (aka DAG)
- Directed cyclic graph
- Undirected acyclic graph
- Undirected cyclic graph

Each type has its own use cases, which we are not going to get into. Just be aware that there are different kinds of graphs and the difference is significant regarding how you create and analyze them.



https://stackoverflow.com/q/20556802/773376

Common graph data structures

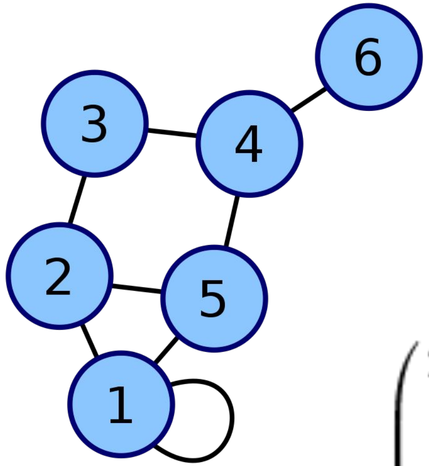
An **adjacency matrix** is perfect for working with smaller graphs as the search/insert/delete time is *O(1)*, and size is *O(n^2)* for directed graphs and undirected graphs it is *O(n^2/2)*.

In essence, one dimension of the matrix represents “from” vertices and the other “to” vertices, and the addressable intersection contains a 1 if the edge exists in that direction, or a 2 if a vertex links back to itself.

Undirected graphs are a mirror along the diagonal (edge 2-3 is the same as edge 3-2), hence only ½ of the matrix needs to be represented.

There are some interesting mathematical properties that arise from this representation that are based in linear algebra (which we are not going to delve into).

https://en.wikipedia.org/wiki/Adjacency_matrix



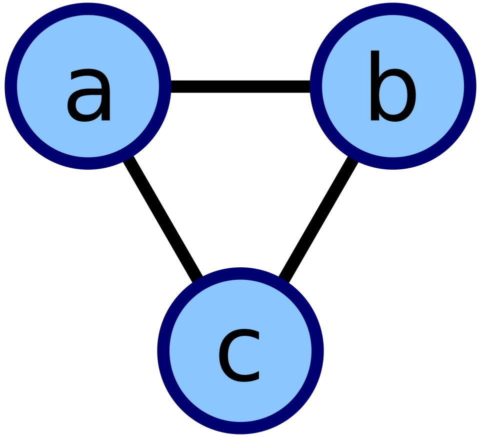
$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Common graph data structures

An **adjacency list** is a list of lists that is better suited to large and/or sparse graphs than an adjacency matrix as its size is dependent only on the number of vertices and edges without wasting memory for unused potential edges. While not as fast in search/update/delete, the operations are still polynomial time in terms of the number of vertices and nodes.

https://en.wikipedia.org/wiki/Adjacency_list

A variant of adjacency lists, called an **edge list**, is shown here.



This undirected cyclic graph can be described by the three unordered lists {b, c}, {a, c}, {a, b}.

Working with graph data

We are going to leverage a Python package named **NetworkX** that is designed to make construction, manipulation, query, and visualization of graphs fairly straightforward. It is optimized for small-medium sized, in-memory graphs, but is really only limited by your computing hardware. (You’d be surprised how large a graph you can fit in only a few GB of RAM.)

Visualizing graph data

Visualizing graphs is a bit more tricky. NetworkX has some rudimentary support, but it’s not robust when you get to larger graphs and not very visually striking.

Netgraph provides some incredibly advanced visualizations for graphs, but can be slow depending on complexity and graph type.

Depending on your specific use case, there are often many other visualization packages that can provide very specific visualizations and/or infographics from raw data.

Graph datasets

There are not quite as many graph datasets out there as there are tabular datasets, and many are from the bioinformatics domain (protein/molecule structures, neural pathways, etc). A few open research dataset sites are:

- https://graphdatasets.com/index.php
- https://graphchallenge.mit.edu/data-sets
- https://ogb.stanford.edu/

Most of these are in either some CSV/TSV adjacency/edge list format, or in MatrixMarket format, which can be read directly into NetworkX with a SciPy helper:

https://networkx.org/documentation/stable/reference/readwrite/matrix_market.html

Graph datasets

Other interesting sources are social networks, where you can often download your personal social network in a format supported (directly or indirectly) by NetworkX:

https://duckduckgo.com/?q=export+social+network+for+networkx

Natural Language Processing

Say what?

Drinking from a firehose

Natural Language Processing (and its sibling **Natural Language Understanding**) is a massive topic. There are whole courses, books, and research centers where the only focus is on trying to comprehend unstructured text or spoken content.

NLP is an important skill to have if you plan on working with anything not available in a structured format like we have seen so far, and is an active research area that many people find intriguing.

Oh, and it's the basis for the famous Turing Test for intelligence. You might have heard of it?

Case in point - the slide title above, are we *literally* drinking from a firehose in this section, or figuratively? How confusing could that phrase be for a non-native English speaker? Or even for a native speaker who has never heard it before? Now imagine a chatbot or virtual assistant trying to make sense of it!

What is Natural Language Processing?

Natural language processing (NLP) is the field that involves the actual processing of textual data so that it can be used by computer systems. It is responsible for **tokenizing** and **parsing** text, often providing **annotations** for **parts of speech (POS)** and **co-references** for **pronouns** and nonspecific **objects**. It tries to identify **entities** (e.g. is the "White House" a proper name or simply a house that is white?).

Some subfields involve data mining, knowledge base construction, and language modeling; there are many more highly specialized fields in NLP that we won't be exploring here. Suffice to say it is a very rich field of research, and incredibly active.

NLP is required to preprocess textual data (corpora) for both heuristic and ML/DL approaches, with the exception of character/grapheme-based models.

(Not to be confused with Neuro-Linguistic Programming which is a whole other beast...)

What is Natural Language Understanding?

Natural language understanding (NLU) on the other hand is a research area with heavy overlap in NLP as it requires the ability to process textual data, but it has its own research focus on trying to **understand** what the text contains at a **conceptual level** such as **intent**, **context**, and **semantics**.

The “understanding” part is where most language-driven systems break down, and what is crucial to creating useful natural language systems that can grasp the same concepts that humans can.

We’re primarily going to focus on NLP for this section as NLU is a very active research area, and in spite of all the hype surrounding “foundational language models” they don’t really “understand” - at best they match common linguistic patterns.

Why is it important?

- People think in their native language, not hexadecimal or l33t. (Ok, most people.)
- Most content out there is still in non-computer-friendly (aka “unstructured”) formats
  - We’re still a long way from a truly semantic web
  - Competing “standards” and incomplete implementations
  - Legacy content!
- It’s a natural method of interaction/socialization for humans
  - Ask questions (learning)
  - Storytelling (knowledge transfer)
  - Aggregate information (knowledge archival)

Quick PSA re NLP biases/dangers

- Most digital text corpora are in English, followed by Mandarin Chinese, with most other languages languishing; there is a language bias in many corpora and LMs
  - https://thegradient.pub/the-benderrule-on-naming-the-languages-we-study-and-why-it-matters/
- Corpora age; older material has dated biases
  - king/queen ⇄ man/woman ⇄ doctor/nurse
- Corpora source; domains are not always cross-compatible
  - “How do I fix a broken collar?”
- Personally identifiable information (PII) leakage
  - Dataset: “Alice Roberts suffers from depression and has signs of bipolar disorder.”
    Q&A model: “Does Alice Roberts suffer from anything?”
    Completion model: “Alice Roberts suffers from…”

What do text datasets look like?

Unlike tabular or graph data, text datasets are pretty much - text. Also, they are typically referred to as corpora (collection of text samples) or a corpus (a single text sample).

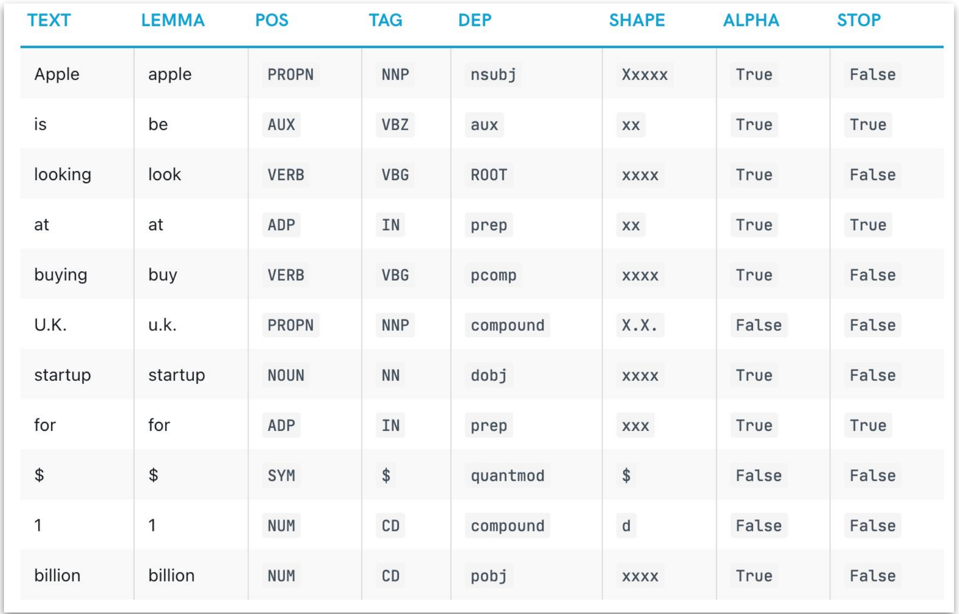
The most basic kind of corpora is **plain text**, just a file of words like you would type in Notepad or the source code for your last project. Due to a lack of any additional information, there is a limit to what can be done with plain text corpora. That being said, simple language models and parsers can be created from plain text, which can provide tools for analyzing new text as well as generating surprisingly interesting content based on the training data.

What do text datasets look like?

Annotated corpora provide not only the text, but also annotations (often made by humans, but some are automated) that can provide a plethora of information depending on which annotation protocol they follow.

This example from spaCy shows several annotations that the model was trained with.

https://spacy.io/usage/linguistic-features



TEXT	LEMMA	POS	TAG	DEP	SHAPE	ALPHA	STOP
Apple	apple	PROPN	NNP	nsubj	Xxxxx	True	False
is	be	AUX	VBZ	aux	xx	True	True
looking	look	VERB	VBG	ROOT	xxxx	True	False
at	at	ADP	IN	prep	xx	True	True
buying	buy	VERB	VBG	pcomp	xxxx	True	False
U.K.	u.k.	PROPN	NNP	compound	X.X.	False	False
startup	startup	NOUN	NN	dobj	xxxx	True	False
for	for	ADP	IN	prep	xxx	True	True
$	$	SYM	$	quantmod	$	False	False
1	1	NUM	CD	compound	d	False	False
billion	billion	NUM	CD	pobj	xxxx	True	False

What are corpora?

Corpora in latin literally means "body." A corpora of text in the context of NLP is a body of text. Corpora are also often referred to as a **collection**. For example, a magazine is a collection of articles.

A **corpus** is the singular of corpora, and therefore a single *discrete* body of text used for document-level analysis. Depending on your corpora, this might be a whole paper, a paragraph, phrase, or even a sentence. Often we are talking about a **document** in a collection.

Documents are broken down most often into **sentences** for the purposes of NLP. That being said, there are areas of research that focus on summarizing discrete chunks of text, so they will examine content based on **sections** and/or **paragraphs** for their experiments.

Sentences are commonly broken down into **tokens**. These are the basic linguistic building blocks of communication, most often words or **named entities**.

In the extreme, some systems will further subdivide tokens into **graphemes**, which sometimes are required to capture the finer points in a token, depending on the language.

Natural Language Toolkit

The **Natural Language Toolkit** (NLTK) is a foundational framework for NLP. It allows for low-level processing of corpora as well as higher-level abstractions such as statistical language models. It can leverage just about any kind of content, be it plain text or annotated content.

NLTK provides parsers, tokenizers, stemmers, part-of-speech tagging, and a host of other NLP tools.

It’s not particularly performant, but it does provide a great base for NLP research and is used quite often as part of a preprocessing pipeline.

Natural Language Processing with Python is a great introductory text to NLP core concepts and how to apply the NLTK package to NLP use cases.

GenSim

The **GenSim** toolkit provides many document tagging, frequency analysis, and word embedding algorithms in a performant and straightforward package for the general research area referred to as “topic modeling.”

Where NLTK can process the corpora into usable sentences and tokens, GenSim can then use these to analyze term frequencies, develop fingerprinting for documents, and create word embeddings for use in statistical models.

If constructed properly, in theory word embeddings are able to mathematically associate the usage of words with each other as observed in the training corpus:

[king] - [man] + [woman] = [queen]

This makes word vectors very popular as a feature engineering approach for ML inputs.

spaCy

“spaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython. The library is published under the MIT license and its main developers are Matthew Honnibal and Ines Montani, the founders of the software company Explosion.

Unlike NLTK, which is widely used for teaching and research, spaCy focuses on providing software for production usage.”

– https://en.wikipedia.org/wiki/SpaCy

NLP Reference Material

The Stanford NLP Group

Jurafsky and Manning @ Stanford are at the bleeding edge of NLP research. They produce parsers, linguistic analyzers, NLP datasets, and are the home of tools like DeepDive and Snorkel for knowledge discovery and data mining (KDD) of unstructured text - and that's just scratching the surface. They are also the home to the SQuAD project which provides a question-answer dataset, and manages a leaderboard of Q&A systems that are tested against the SQuAD dataset.

The latest draft of their canonical NLP textbook is available for free online: Speech and Language Processing, 3e (Draft).

NLP Reference Material

The NLP Index

Index site for NLP research datasets with metadata and links to papers where available.

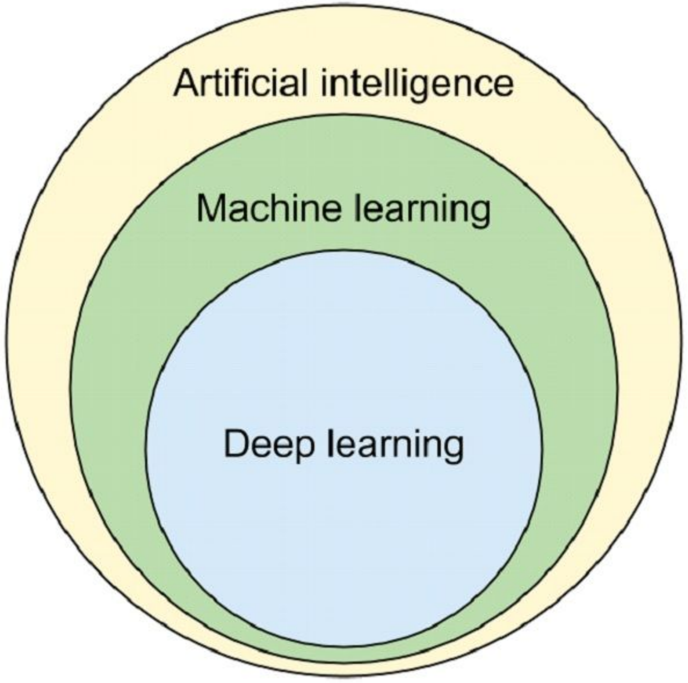
Papers With Code

“The mission of Papers with Code is to create a free and open resource with Machine Learning papers, code, datasets, methods and evaluation tables. We believe this is best done together with the community, supported by NLP and ML.”

Intro to Machine Learning

How does it know what it doesn’t know?

What is Machine Learning?



“Machine learning is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead.”

https://en.wikipedia.org/wiki/Machine_learning

Machine Learning (ML) is a subfield within the broader Artificial Intelligence area of research.

Note that **Deep Learning** (DL) is a subfield within ML.

What is Machine Learning?

… aka “**statistical machine learning**” as it’s mathematically based, not knowledge based; systems optimize a function approximation.

Part of the “connectionist” movement in AI as opposed to “symbolic” knowledge-based AI like expert systems, heuristics, case-based reasoning, BDI, etc.

Can be used for **classification** (e.g. dog or cat?) or **regression** (e.g. pricing forecasts, population size, etc) inference/prediction.

Can be used to identify **latent grouping structures** in the data via **clustering**.

Great at **“pattern matching”;** should not be confused with “**intelligence.**” 🙈🙉🙊

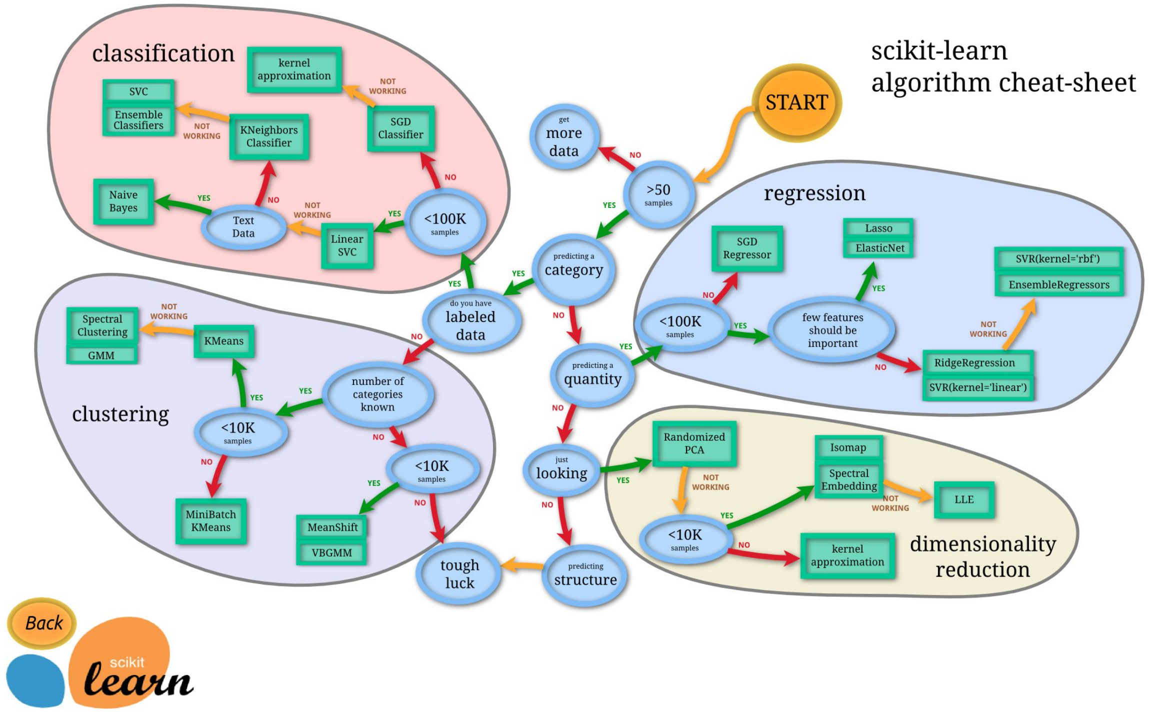
No matter how impressive or huge the hype, still considered a “weak” form of AI as opposed to Artificial General Intelligence (AGI) which only exists in sci-fi.

What is Machine Learning?

While commonly associated with artificial neural networks (ANNs) and deep learning (DL) models, ML as a field has an incredible breadth of algorithms available and under active research.

While a bit dated, this workflow from Scikit-learn attempted to map out how to select one of the many ML algorithms they provided based on meta-level criteria such as required functionality, number of features, and size of datasets to name a few.

Ultimately, the best ML model is the **simplest** and most **explainable**, **understandable** and **transparent** one that achieves the **target accuracy** and **generalizes well** for the use case over unseen data.



https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

ML terminology

Model/Estimator: The collection of parameters that capture the function approximation for a given machine learning algorithm learned through training.

Function approximation: Because many hypothesis spaces for an ML model are so large, we cannot usually get "the right answer" so the ML algorithm learns the approximate function that gives the closest answer possible.

Classification: Model type that predicts a label for a sample, from a finite set of labels (e.g. cat vs dog).

Regression: Model type that predicts continuous values for a sample (e.g. demand-driven pricing).

Training data(set): Data used to train an ML model.

Validation data(set): Data used to validate the model during training, used to inform the optimization function. Ideally should be different from the training set; in smaller datasets, this can be a challenge.

Test data(set): Data used to test a trained model (one with maximum accuracy and minimum error). The model is ideally never trained with this data so that it can be tested against unseen samples. In smaller datasets, this can be a challenge.

Features: The sample/observation data you want the algorithm to be able to make predictions on. Also often referred to as "X" in ML algorithms.

Label/target: In supervised learning, the desired output for inference/prediction on a sample. Also often referred to as "Y" in ML algorithms.

Optimization function: Many ML algorithms actively maximize accuracy and minimize error; there are several approaches, and these are classed as optimization functions. Most common is Stochastic Gradient Descent (SGD) and its derivatives. Often part of model tuning.

Activation function/kernel function: A function, often differentiable, used by ML algorithms to non-linearly transform data within the model, either to simplify the model or project data into higher dimensions. Often part of model tuning.

Accuracy: How "correct" the model is when predictions are compared against ground truth. Depending on the type of model, there are several different accuracy measurements that can be used.

Error/Loss: How far off the model is from the ground truth. Depending on the type of model, there are several different error measurements that can be used. Used to guide optimization updates to models

For a much more comprehensive glossary, check out:
https://www.analyticsvidhya.com/glossary-of-common-statistics-and-machine-learning-terms/

High level classes of ML algorithms

- Supervised Learning
  - Requires labeled training data
- Unsupervised Learning
  - Does not require labeled data, but can leverage it if available
- Reinforcement Learning
  - Requires an agent, environment, and reward system
- Self-Supervised Learning
  - Hybrid SL and UL approach, newer way to tackle large training datasets that are mostly unlabeled

Why Supervised Learning?

It is by far the most prevalent form of ML used; a non-exhaustive list of examples:

- Price/cost predictions
    - Uber, Lyft demand-based pricing models
    - Quantitative predictions in finance
- Computer Vision (CV)
    - Image classification
    - Object detection
    - Image segmentation
    - Distance estimation, “range-finding”
- Natural Language Processing
    - Sentiment classification - e.g. detecting hate speech
    - Intent classification - e.g. chatbot script drivers
    - Translation engines
    - Spam detectors
- Core algorithm for Generative Adversarial Networks (GANs)
- Core algorithm for DRL approaches
- Etc ad nauseum…

Why Supervised Learning?

It’s also the easiest to get started with:

- Tons of toy datasets
- Material/examples are widely available
- Smaller models can easily be trained/tested on consumer-grade compute resources in a reasonable amount of time
- Consistent interfaces within Scikit-learn for experimentation/proof-of-concept

The groundwork will prepare you for concepts used in the other ML subfields

Caveats

- Requires labeled data
    - Lots of academic/research datasets available
        - Need to be aware of source, age, and domain as they can introduce unwanted bias
        - Ensure provenance/quality of data
    - Large datasets with good labels are harder to find (“data is the new `<oil | gold | currency>`”)
    - Sometimes a hybrid approach (e.g. self-supervised) can help with labeling, but can introduce more error/bias depending on the approach used
- More features == more data == more training
    - Universal to any SL approach (“curse of dimensionality”), but can get exponentially more complex using ANNs in particular
    - Feature engineering/transforms/dimensionality reduction approaches can often address this, but sometimes at a cost (time complexity, loss in accuracy/generalization)

https://live.staticflickr.com/3875/14998434959_6b726eb200.jpg

Scikit-learn has your back!

You're not going to have to (nor should you need to) implement any of these algorithms from scratch unless you are:

1. Building a new ML framework in a new ecosystem (while a great way to learn the algorithms, is a non-trivial task).
2. Involved in ML R&D - as in developing new models, layers and/or architectures, not just constructing and training models from existing libraries.
3. I'm positive there's a #3…

The **Scikit-learn** package (along with NumPy, SciPy and Pandas) have totally got you covered for anything short of complex DL models and computer vision (CV).

There are even documented instances of Scikit-learn models in production. 😎

Scikit-learn has your back!

Many common ML algorithms (and their variants) are already packaged up in classes making them reasonably easy to train and test ML models, build ML pipelines, run “grid search” experiments, etc.

Bayesian networks	sklearn.naive_bayes: Naive Bayes
Decision trees	sklearn.tree: Decision Trees
Support Vector Machines	sklearn.svm: Support Vector Machines
Ensembles	sklearn.ensemble: Ensemble Methods
K-Nearest Neighbors (kNN)	sklearn.neighbors: Nearest Neighbors
Artificial neural networks	sklearn.neural_network: Neural network models

Scikit-learn architecture

Scikit-learn is **object-oriented**, and there are **three major classes** of algorithms:

- Classifiers
  - Algorithms that are used for classification problems (e.g. dog or cat)
- Regressors
  - Algorithms that are used for regression problems (e.g. price prediction)
- Transformers
  - Algorithms that transform data for a plethora of use cases:
    - Dimensionality reduction
    - Scaling
    - Reshaping
    - Etc

Scikit-learn architecture

The package has a fairly uniform API:

Classifiers and Regressors (aka Estimators)

- *fit(x, y)*
  - Trains the model, where x is your features and y is your labels
- *predict(x)*
  - Predicts y for the given x on a trained mode
- *score(x, y)*
  - Score the accuracy of you model using the provided x, y data; default metric is R2

Transformers

- *fit(x[, y])*
  - “Train” a transformer on provided x features (and optional y labels; depends on the algorithm)
- *transform(x)*
  - Perform the transform on the provided x features
- *fit_transform(x[, y])*
  - Combine the fit() and transform() functions into one call for a one-shot transformation

Intro to Deep Learning

Gaze into the abyss…

What is Deep Learning (DL)?

“When you hear the term deep learning, just think of a large deep neural net. Deep refers to the number of layers typically and so this kind of the popular term that’s been adopted in the press. I think of them as deep neural networks generally.”

-- Jeff Dean, Google

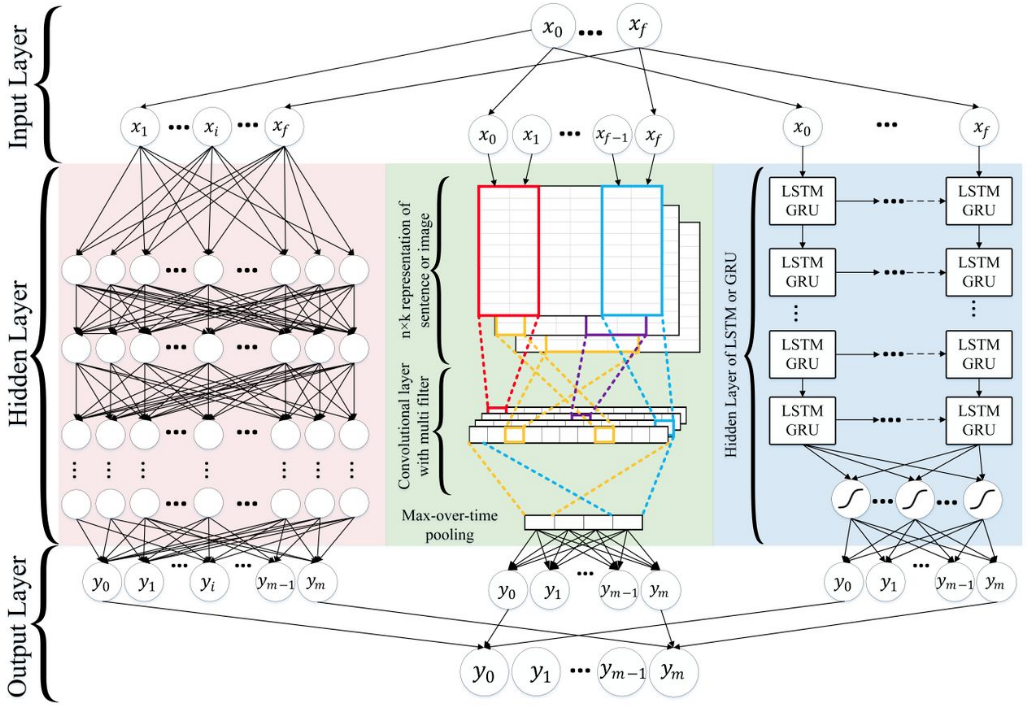
The term’s origin is often attributed to Geoffrey Hinton’s paper “A Fast Learning Algorithm for Deep Belief Nets” (https://doi.org/10.1162/neco.2006.18.7.1527)

In industry, usually refers to a network with 2+ hidden layers - so not always *that* deep… #marketecture 😒

What is Deep Learning (DL)?

This is what most people visualize when someone mentions DL. From an ANN with two hidden layers to something this large, researchers started small and built the model layer by layer, testing each to ensure it was improving accuracy and reducing error while generalizing better.

(Meanwhile, the more complex these models get, the less explainable they become and you hear the term “black box” or “opaque box” used to describe them.)



Input Layer
Hidden Layer
Output Layer
x_0 ... x_f
x_1 ... x_i ... x_f
x_0 x_1 ... x_{f-1} x_f
x_0 ... x_f
n×k representation of sentence or image
Convolutional layer with multi filter
Max-over-time pooling
Hidden Layer of LSTM or GRU
LSTM GRU
y_0 y_1 ... y_i ... y_{m-1} y_m
y_0 y_1 ... y_{m-1} y_m

https://commons.wikimedia.org/wiki/File:Random_Multimodel_Deep_Learning_(RMDL).png

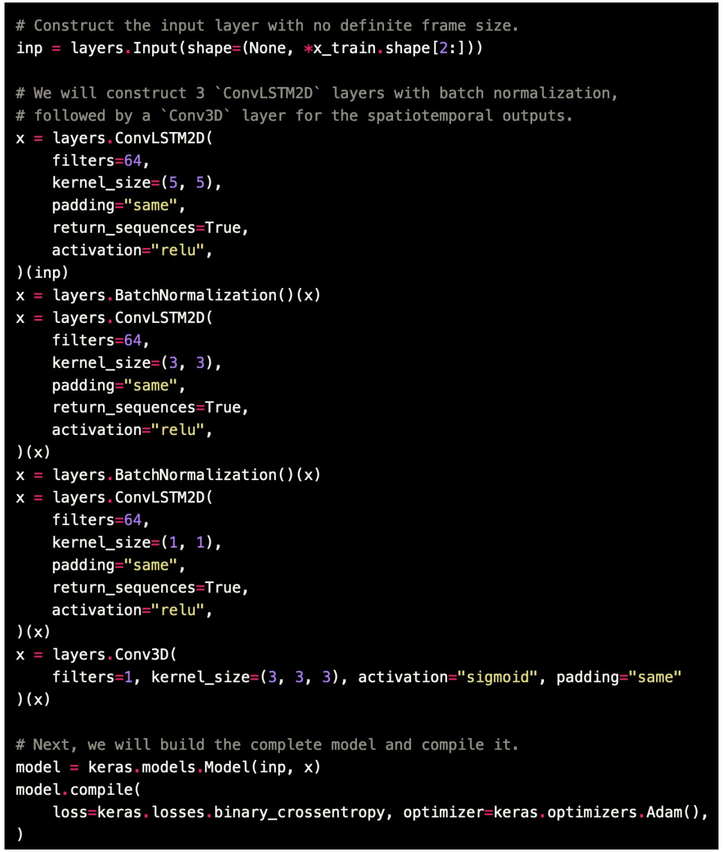
What are these “layers” you speak of…?

In an ANN, a layer in its simplest form accepts **upstream inputs**, applies some **transformation** to them to extract patterns for **downstream use**, and passes those **outputs** to the next layer. You can think of a DL model as a kind of data processing pipeline. They are often represented internally as a computational directed graph where the vertices represent the transform layers and the edges represent the flow of data.

Layers can be as simple as an **Input** layer (initial layer in a model, accepts your sample features as a starting point for feeding forward) or as complex as a **Convolutional 2D Long-Short Term Memory (ConvLSTM2D)** with multiple tuning hyperparameters and extensive input and output options for handling long sequences of data. Still others can apply arbitrary transformations on your data like **BatchNormalization** or reshape it for a downstream layer with specific requirements (e.g. between convolutional and dense layers).

Example layers and functional model construction code for video frame prediction model in TensorFlow Keras sourced from:

https://keras.io/examples/vision/conv_lstm/#model-construction



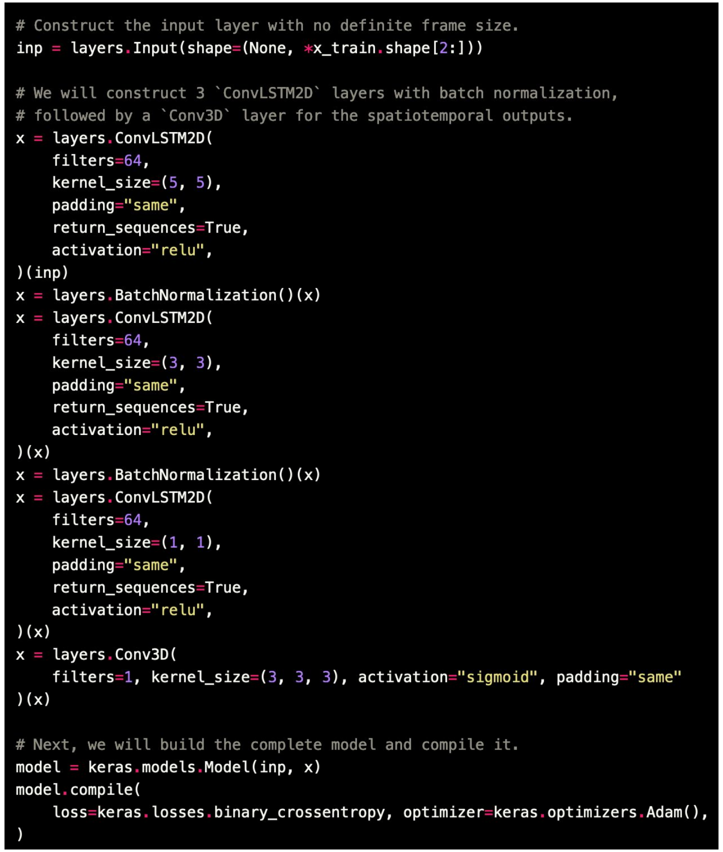
```python
# Construct the input layer with no definite frame size.
inp = layers.Input(shape=(None, *x_train.shape[2:]))

# We will construct 3 `ConvLSTM2D` layers with batch normalization,
# followed by a `Conv3D` layer for the spatiotemporal outputs.
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(5, 5),
    padding="same",
    return_sequences=True,
    activation="relu",
)(inp)
x = layers.BatchNormalization()(x)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(3, 3),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
x = layers.BatchNormalization()(x)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(1, 1),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
x = layers.Conv3D(
    filters=1, kernel_size=(3, 3, 3), activation="sigmoid", padding="same"
)(x)

# Next, we will build the complete model and compile it.
model = keras.models.Model(inp, x)
model.compile(
    loss=keras.losses.binary_crossentropy, optimizer=keras.optimizers.Adam(),
)
```

What are these “layers” you speak of…?

Note that even though this model “only” has seven layers, it is considered a DL network.

Most DL-capable frameworks (e.g. PyTorch, TensorFlow) provide ready-to-use implementations of popular (and sometimes more esoteric) layers for you to mix into your network where needed.

Also note the last line where the loss (aka error) and optimizer are specified at model compile time (Adam is an adaptive variant of SGD mentioned earlier). These are used for evaluating the model during training (loss) and update the model to reduce the error (optimizer).



```python
# Construct the input layer with no definite frame size.
inp = layers.Input(shape=(None, *x_train.shape[2:]))

# We will construct 3 `ConvLSTM2D` layers with batch normalization,
# followed by a `Conv3D` layer for the spatiotemporal outputs.
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(5, 5),
    padding="same",
    return_sequences=True,
    activation="relu",
)(inp)
x = layers.BatchNormalization()(x)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(3, 3),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
x = layers.BatchNormalization()(x)
x = layers.ConvLSTM2D(
    filters=64,
    kernel_size=(1, 1),
    padding="same",
    return_sequences=True,
    activation="relu",
)(x)
x = layers.Conv3D(
    filters=1, kernel_size=(3, 3, 3), activation="sigmoid", padding="same"
)(x)

# Next, we will build the complete model and compile it.
model = keras.models.Model(inp, x)
model.compile(
    loss=keras.losses.binary_crossentropy, optimizer=keras.optimizers.Adam(),
)
```

The DL landscape

There are a plethora of DL frameworks out there, in multiple language ecosystems - but there are two major players in Python OSS land.

TensorFlow (Google) gained rapid popularity when it was first released in 2015, and for a while was the go-to for approachable DL modeling, especially with the introduction of the **Keras** abstractions. It has since branched into model serving, embedded/edge, and the JAX compute backend is being leveraged outside of TF for other compute-intensive applications.

PyTorch (Facebook AI Research) followed quickly in 2016, and was quickly adopted in academic and industry research due to its sensible abstractions of the DL training workflow and ease with which new layers and model architectures could be rapidly experimented with and deployed. It has since become more mainstream with the advent of tools like **Lightning** and early successes in edge/IoT environments. These days it also tends to be a bit faster on more complex models than TF due to a highly-optimized compute backend.

TensorFlow user
You have StableDiffusion now?
Pytorch user
Do you mean the community fork or the official one?
JAX user
You guys have community forks?
MXnet user
You guys have a community?

Thank you! Questions?

arpieb/adc2022-foss-ds-ml-tools